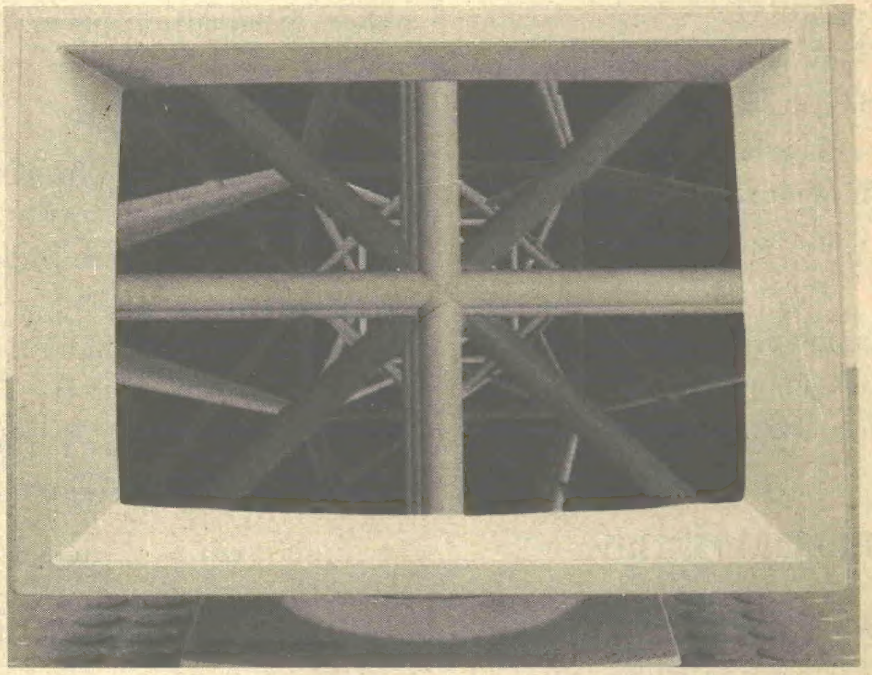
การประยุกต์ C++ ใช้กับงานเวกเตอร์ : การบวก ลบ และคูณเวกเตอร์



กล่าวถึงการประยุกต์ C++ เพื่อใช้กับงานทางด้านคณิตศาสตร์ที่เกี่ยวกับเวกเตอร์

โอเปอเรเตอร์ที่ใช้กับการบวก ลบ นิเสธ และการคูณ จะเขียนด้วยคำสั่งต่อไปนี้

```
friend vector operator + (const vector & v, const & w);
friend vector operator = (const vector & v, const & w);
friend vector operator = (const vector & v);
friend vector operator * (double a, const vector & w);
friend vector operator * (const vector & v, double b);
friend double operator * (const vector & v, const & w);
```

คืนค่าสเกลาร์

โอเปอเรเตอร์ที่ทำหน้าที่เป็นผลคูณสเกลาร์จะคืนค่าเป็นสเกลาร์ชนิด double ดังนี้

โอเปอเรเตอร์เหล่านี้เข้าใจวิธีการคำนวณจากข้อมูลที่มีอยู่ได้ดีกว่าการส่งข่าวสารไปยังออปเจ็กต์เฉพาะ จึงสมควรประกาศให้เป็นเฟรนด์แทนการประกาศการเป็นสมาชิกของคลาสเวกเตอร์

โอเปอแรนด์เวกเตอร์ทั้งหมดจะถูกส่งผ่านโดยการอ้างอิง (เพื่อหวังผลในแง่ประสิทธิภาพ) การผ่านโดยอาศัยค่าจำเป็นที่จะต้องก๊อบปี้แต่ละโอเปอแรนด์ขึ้นใหม่ทุก ๆ ครั้งที่โอเปอเรเตอร์ถูกเรียกใช้

คีย์เวิร์ด const ช่วยกัน หนทางที่ทำให้โอเปอเรเตอร์เปลี่ยนค่าของโอเปอแรนด์

โอเปอแรนด์ซึ่งคืนผลลัพธ์ในรูปของเวกเตอร์จะคืนโดยวิธี value ถึงแม้การคืนค่าโดยวิธีอ้างอิง (reference) จะมีประสิทธิภาพมากกว่าก็ตาม

เหตุผลก็คือ แต่ละนิยามที่สมนัย

กันกับการประกาศเวกเตอร์ท้องถิ่น u จะเก็บผลลัพธ์ไว้ แต่วัตถุท้องถิ่นจะถูกทำลายขณะที่ที่ฟังก์ชันคืนค่าให้ ซึ่งถ้าจะใช้การคืนค่าแบบอ้างอิงกับออปเจ็กต์ท้องถิ่นก็จะไม่มีความหมายอันใด

นิยามของโอเปอเรเตอร์เหล่านี้คล้ายกัน ดังนั้นเราจึงมองดูรายละเอียดเฉพาะนิยามโอเปอเรเตอร์บวกก็คงเพียงพอ (ดูรูปที่ 1)

ซึ่งเวกเตอร์ u ถูกประกาศเพื่อจะเก็บค่าผลบวกของเวกเตอร์ 2 เวกเตอร์ที่กำหนดให้

เนื่องจาก operator+ เป็นเฟรนด์มากกว่าที่จะเป็นสมาชิกของคลาสเวกเตอร์ เราจึงต้องใช้ชื่อที่ได้แก้ไขไว้แล้วอ้างถึงตัวแปรอินสแตนซ์ของเวกเตอร์ u, v และ w

คำสั่ง for กำหนดตัวแปรอินสแตนซ์แต่ละตัว u.c[i] ของ u เก็บค่าผลบวกตัวแปรอินสแตนซ์ v.c[i] ของ v และ w.c[i] ของ w

```
vector u;
for (int i = o; i<SIZE; i++)
        u.c[i] = v.c[i] + w.c[i] ;
return u ;
```

รูปที่ 1
รายละเอียดของนิยามโอเปอเรเตอร์บวก

```
for (int i = o; i<SIZE;i++)
        if (v.c[i] !=w.c[i]) return o;
return 1;
```

รูปที่ 2
นิยามของโอเปอเรเตอร์ = =

โอเปอเรเตอร์เท่ากัน (Equality Operators)

เวกเตอร์สองเวกเตอร์เท่ากัน ก็ต่อเมื่อ Components ที่สมนัยกันเท่ากัน เพื่อจะให้ผู้ใช้ได้ทดสอบการเท่ากันของเวกเตอร์ เราจึงสร้างนิยามของโอเปอเรเตอร์เท่ากัน (= =) และไม่เท่ากัน (!=)

```
friend int operator == (const vector & v, const vector & w);
friend ind operator!= (const vector & v, const vector & w);
```

โอเปอเรเตอร์ = = จะให้ค่า 0 (false) ถ้าโอเปอแรนด์ทั้งสองเท่ากัน และ 1 (true) ถ้าโอเปอแรนด์ทั้งสองไม่เท่ากัน

ส่วนโอเปอเรเตอร์ != ให้ผลลัพธ์ตรงกันข้ามกับข้างต้นทั้งสองกรณี ตามปกติโอเปอแรนด์เวกเตอร์ถูกส่งผ่านโดยการอ้างอิง (เพื่อประสิทธิภาพ) ส่วนนิยามของ = = นั้นตรงไปตรงมาดังรูปที่ 2

ส่วนนิยามของ != ก็เปลี่ยนแปลงเฉพาะค่าที่ส่งกลับ เช่น

ในนิยามของ = = return 0; ก็เปลี่ยนเป็น return 1;

และ return 1; ก็เปลี่ยนเป็น return 0; เป็นต้น

โอเปอเรเตอร์กำหนดค่า (Assignment Operators)

ดังที่ได้กล่าวมาแล้ว เราไม่ได้นิยามโอเปอเรเตอร์กำหนดค่า (=) ขึ้นมาใหม่ เพียงแต่ใช้การกำหนดค่า memberwise โดยปริยายอย่างไรก็ตามเราจะกำหนดโอเปอเรเตอร์สามตัวที่นำเอา = มาใช้รวมกับโอเปอเรชันตัวอื่น ๆ ได้ เช่น

```
vector & operator + = (const vector & w);
vector & operator - = (const vector & w);
vector & operator * = (double b);
```

ซึ่งโอเปอเรเตอร์ข้างต้นนำมาใช้ได้ดังนี้

```
v + = w; // add w to v
v - = w; // subtract w from v
v *= 2; // multiply v by 2
```

จะเห็นว่าการคูณสองแบบคือ (2*v และ v*w) ไม่สามารถจะใช้การกำหนดค่าลูกผสมได้ เพราะชนิดของข้อมูลต่างกันกับโอเปอแรนด์แรกนั่นเอง เนื่องจากโอเปอเรเตอร์กำหนดค่าแต่ละตัวเป็นการเปลี่ยนค่าแรกของโอเปอแรนด์ เราจึงคิดถึงโอเปอเรเตอร์กำหนดค่าเหมือนกับการส่งข่าวสารไปยังโอเปอแรนด์แรกนั่นเอง ดังนั้นเราจึงประกาศโอเปอเรเตอร์กำหนดค่าให้เป็นสมาชิก แทนที่จะเป็นเฟรนด์

จากการที่ถกกันข้างต้นโอเปอแรนด์ตัวแรกของโอเปอเรเตอร์สมาชิกเป็นออปเจ็กต์ที่โอเปอเรเตอร์จะถูกประยุกต์นำมาใช้ได้ เช่น V+=W;

จะเทียบเท่ากับการเรียกฟังก์ชันดังนี้

```
V.operator + = (w);
```

ดังนั้นโอเปอแรนด์แรกไม่จำเป็นต้องถูกประกาศให้เป็นอาร์กิวเมนต์สำหรับฟังก์ชันโอเปอเรเตอร์ ซึ่งจะเห็นว่าคีย์เวอร์ด const ในการประกาศ เพื่อประกันว่าค่าของโอเปอแรนด์ตัวที่สองจะไม่เปลี่ยนแปลงค่าด้วยการกำหนดค่า ส่วนค่าของโอเปอแรนด์แรกมักจะเปลี่ยนแปลงไปตามสถานการณ์

นิยามฟังก์ชันโอเปอเรเตอร์

ในการนิยามฟังก์ชันโอเปอเรเตอร์ เราใช้ชื่อที่ไม่ได้รับการแก้ไขอ้างอิงถึงตัวแปรอินสแตนซ์ของโอเปอแรนด์ตัวแรก แต่เราต้องใช้ชื่อที่แก้ไขแล้วอ้างถึงตัวแปรอินสแตนซ์ตัวที่สอง (ถ้าเป็นออปเจ็กต์ของคลาส) ดังนั้นรหัส operator + =, c จะอ้างถึงอะเรย์ของ component สำหรับโอเปอแรนด์ตัวแรก และ w.c จะอ้างถึงอะเรย์ของ component สำหรับโอเปอแรนด์ตัวที่สอง (ดูรูปที่ 3)

ตัวแปรที่ได้นิยามไว้ก่อนแล้ว this เป็นพอยน์เตอร์ที่ชี้ไปยังวัตถุซึ่งฟังก์ชันสมาชิกหรือฟังก์ชันโอเปอเรเตอร์ถูกประยุกต์ใช้ ดังนั้นสำหรับโอเปอเรเตอร์, This ต้องชี้ไปที่โอเปอแรนด์ตัวแรกเสมอ ส่วนโอเปอเรเตอร์กำหนดค่าก็จะคืนค่าของตัวกำหนดค่า (ค่าที่กำหนดให้แก่โอเปอแรนด์ตัวแรก) เนื่องจาก *this เป็นชื่อสมมติสำหรับโอเปอแรนด์ตัวแรก ดังนั้นประโยค

```
return * this;
```

จึงคืนค่าผลลัพธ์ที่ต้องการมาให้ ลองดูการประกาศของโอเปอเรเตอร์กำหนดค่าซึ่งผลลัพธ์โดยวิธีอ้างอิง และโอเปอแรนด์แรกไม่เป็นออปเจ็กต์ท้องถิ่น ตัวอ้างอิงที่มีความหมายจึงถูกคืนมาให้ ดังนั้นเราสามารถใช้สิ่งนี้ได้ด้วยวิธีที่มีประสิทธิภาพเพื่อคืนผลลัพธ์ให้แก่การกำหนดค่า

โอเปอเรเตอร์สับสคริปต์

เราเข้าถึง components แต่ละตัวของเวกเตอร์โดยการใช้โอเวอร์โหลดโอเปอเรเตอร์สับสคริปต์ ,[ ] ดังนี้

```
double & operator [ ] (int i)
```

เรานิยามโอเปอเรเตอร์ [ ] ให้เป็นฟังก์ชันสมาชิก ซึ่ง Operator [ ] จะใช้กับเวกเตอร์ที่ถูกสับสคริปต์ โดยมีอาร์กิวเมนต์ i เป็นตัวสับสคริปต์ เช่น

V[2] จะเหมือนกับการเรียกฟังก์ชัน v.operator [ ] (2) ในทั้งสองแบบนิพจน์คืนค่าอ้างอิงแก่ component ของเวกเตอร์นั่นก็คือเป็นการอ้างอิงแก่สมาชิกของอะเรย์ component c การอ้างอิงที่เป็นค่าทางซ้าย (lvalue) จะสามารถถูกนำมาใช้ได้การรับค่าของ component และการกำหนดค่าใหม่ให้แก่มัน ซึ่งจำนวน component จะหาได้จาก LOWER และ UPPER

ซึ่ง V[LOWER] จะเป็น component ตัวแรกของเวกเตอร์ V และ V [UPPER] จะเป็น component ตัวสุดท้าย

รหัสสำหรับ operator[ ] จะเป็นดังรูปที่ 4

ถ้าสับสคริปต์อยู่นอกช่วง (out of rauge) ก็จะพิมพ์ผลลัพธ์ถึงความผิดพลาด แล้วโปรแกรมก็จะหยุดชะงัก มิฉะนั้นการอ้างถึง C [i−LOWER] จะให้ผลลัพธ์ออกมา นิพจน์ i−LOWER เปลี่ยนสับสคริปต์จากช่วงที่กำหนดขึ้น

```
for (int i=o; i<SIZE;i++)
        c[i] += w.c [i];
return I*this;
```

รูปที่ 3
การอ้างถึงอะเรย์ของโอเปอเรเตอร์ + =,C และ W.C ใน component

```
if (i<LOWER || i> UPPER) {
        cout << "Subscript out of rauge/n";
        exit (1);
}
return C [i-LOWER];
```

รูปที่ 4
รหัสสำหรับโอเปอเรเตอร์ [ ]

สำหรับ operator[ ] (LOWER ไปยัง UPPER) ไปยังช่วงที่ต้องการด้วย C (0 ถึง SIZE −1)

เนื่องจากสมาชิกของ c ถูกคืนค่าโดยการอ้างอิง การกำหนดค่าแก่ v[k] มีผลในการกำหนดค่าแก่ v.c [k−LOWER]

การเปลี่ยนให้เป็นขนาดของสเกลาร์

ในตอนต้นต้นเราได้เห็นวิธีการคอนสตรัคเตอร์ด้วยการนิยามอาร์กิวเมนต์ตัวเดียว เพื่อเปลี่ยนชนิดข้อมูลจากชนิดหนึ่งไปเป็นอีกชนิดหนึ่งโดยอาศัยคลาส เช่น คอนสตรัคเตอร์

```
vector (double*a);
```

เปลี่ยนอะเรย์ a ให้เป็นเวกเตอร์ การเปลี่ยนแปลงเช่นนี้ควรจะเห็นได้ชัดเจน เช่น ถ้า u และ v เป็นเวกเตอร์ และ b เป็นอะเรย์ของข้อมูลชนิด double ดังนั้นนิพจน์ต่อไปนี้ u+b, v−b, u+b+v, เป็นจริง

ในแต่ละกรณีอะเรย์ b จะถูกเปลี่ยนให้เป็นเวกเตอร์ก่อนที่จะบวกหรือลบกับเวกเตอร์อื่น

คอนสตรัคเตอร์จะเป็นการเปลี่ยนออปเจ็กต์ของข้อมูลชนิดอื่นให้เป็นชนิดเดียวกับออปเจ็กต์ของคลาส เราน่าจะสร้างหรือนิยามโอเปอเรเตอร์สำหรับเปลี่ยนออปเจ็กต์ของคลาสให้เป็นชนิดอื่นได้ เช่น โอเปอเรเตอร์ double( ) เปลี่ยนออปเจ็กต์คลาสให้เป็นชนิด double

โอเปอเรเตอร์ int( ) เปลี่ยนออปเจ็กต์คลาสให้เป็น int

เราจะใช้โอเปอเรเตอร์ double( ) เปลี่ยนเวกเตอร์ให้สอดคล้องหรือสมนัยกับขนาดของสเกลาร์ เราคิดถึงเวกเตอร์ในแง่ของทิศทางและขนาด แต่เมื่อเราเปลี่ยนให้เป็นสเกลาร์ก็ต้องคำนึงถึงขนาดเท่านั้น (ตัดทิศทางทิ้งไป) ซึ่งถ้าเปรียบเทียบกันก็คล้าย ๆ กับการเปลี่ยนทศนิยมให้เป็นจำนวนเต็มโดยคงในส่วนที่เป็นจำนวนเต็มของทศนิยมไว้

โอเปอเรเตอร์ double ( ) จะมีหน้าตาดังนี้

```
operator double ( ) const;
```

เนื่องจาก double ( ) เริ่มต้นด้วยตัวอักษร จะมีช่องว่างระหว่างโอเปอเรเตอร์และ double ( ) โอเปอเรเตอร์ถูกประกาศเป็นสมาชิกของคลาสโดยมีโอเปอแรนด์เดี่ยวเป็นออปเจ็กต์ ซึ่งโอเปอเรเตอร์ถูกประยุกต์ใช้และไม่จำเป็นต้องคืนค่าที่บ่งถึงกลับมาตอนประกาศ เพราะว่าค่าที่คืนกลับมาเป็น double ชนิดของโอเปอแรนด์ก็ถูกเปลี่ยนแปลงด้วย

const ที่เป็นตัวบ่งบอกฟังก์ชันสมาชิก operator double ( ) เป็นฟังก์ชันคงตัว มันไม่เปลี่ยนค่าของออปเจ็กต์ซึ่งถูกประยุกต์ใช้

โอเปอเรเตอร์ double ( ) ถูกนิยามดังนี้ (ดูรูปที่ 5)

คงจำกันได้ว่าเราหาขนาดของเวกเตอร์โดยการนำเอา components มายกกำลังสองบวกกันแล้วถอดรากที่สอง ฟังก์ชัน sqrt ( ) ประกาศใช้อยู่ใน math.h

โอเปอเรเตอร์ double ( ) จะใช้ได้ทั้งในฟังก์ชันหรือใน type-Cast ดังนี้

```
double (v)
```

และ

```
(double) v
```

จะแทนขนาดของ v ดังนั้นถ้า v เป็นเวกเตอร์ และ x เป็นตัวแปรชนิด double แล้ว x=v; จะกำหนดค่าซึ่งเป็นขนาดของ v ให้แก่ x

อินพุตและเอาต์พุต

เราจัดให้มีอินพุตและเอาต์พุตของเวกเตอร์โดยการทำโอเวอร์โหลดโอเปอเรเตอร์ >> และ << โดยการประกาศดังนี้

```
friend istream & operator >> (istream ci, vector & v);
friend ostream & operator << (ostream & co, const vector & v);
```

จะเห็นว่าโอเปอแรนด์เวกเตอร์ >> ถูกส่งผ่านโดยวิธีอ้างอิงเพราะว่า >> เปลี่ยนค่าของโอเปอแรนด์ โอเปอแรนด์เวกเตอร์ << จะถูกส่งผ่านโดยวิธีส่งค่าหรือวิธีอ้างอิง เราใช้วิธีอ้างอิงก็เพื่อต้องการประสิทธิภาพ และเราใช้ const เพื่อประกันไม่ให้โอเปอเรเตอร์เปลี่ยนค่าของโอเปอแรนด์

โอเปอเรเตอร์ >> อ่าน SIZE จากสตรีม C แล้วเก็บไว้ในสมาชิกของ v.c: ดูรูปที่ 6

สัญลักษณ์วงเล็บและคอมมา (,) ไม่ใช้สำหรับการอินพุต อินพุตจะประกอบด้วยค่าของ component ที่ต้องการแยกด้วยช่องว่าง

```
vector :: operator double ( ) const
{
        double sum =o.o;
        for (int i =o; i<SIZE; i++)
                sum += c[i] * c[i];
        return sqrt (sum);
}
```

รูปที่ 5
โอเปอเรเตอร์ double ที่ถูกนิยาม

```
for (int i =o; i<SIZE;i++)
        ci >> v.c [i];
return ci;
```

รูปที่ 6
โอเปอเรเตอร์ >> อ่าน SIZE จากสตรีม C แล้วเก็บไว้ในสมาชิกของ V.C:

```
co << "(" << v.c. [o];
for (int i = 1; i<SIZE;i++)
co << "/" << v.c [i];
co << ")";
return co;
```

รูปที่ 7
นิยามของโอเปอเรเตอร์ <<

นิยามของ << ค่อนข้างซับซ้อนกว่า >> เพราะว่า << ใช้สัญลักษณ์วงเล็บและคอมมา ดังนี้ (ดูรูปที่ 7)

ลิสติงที่ 1 vectorc.cpp

```
//File vectorc.cpp
//Source file for class vector
#include <stdlib.h>
#include <math.h>
#include "vectorc.h"

//Constructor:create null vector
vector::vector()
{
 for(int i=0;i<SIZE;i++)
 c[i]=0.0;
}
//Constructor:create vector from array
vector::vector(double* a)
{
 for (int i=0; i<SIZE;i++)
  c[i]=a[i];
}
//vector addition
vector operator+ (const vector&,const vector& w)
{
 vector u;
 for (int i=0; i<SIZE; i++)
  u.c[i]=v.c[i]+w.c[i];
 return u;
}
//Vector subtraction
vector operator- (const vector& v, const vector& w)
{
  vector u;
  for (int i=0;i<SIZE; i++)
    u.c[i]=v.c[i]-w.c[i];
 return u;
}
//VECTOR NEGATION
vector operator- (const vector& v)
{
 vector u;
 for (int i=0;i<SIZE;i++)
  u.c[i]=-v.c[i];
  return u;
 }
 //Left scalar multiplication
 vector operator* (double a,const vector& w)
 {
 vector u;
 for(int i=0;i<SIZE;i++)
  u.c[i]=a*w.c[i];
 return u;
}
//Right scalar multiplication
vector operator* (const vector& v,double b)
{
  vector u;
    for (int i=0;i<SIZE; i++)
      u.c[i]=v.c[i]*b;
    return u;
 }
 //Scalar product
double operator* (const vector& v,const vector& w)
{
```

ลิสติงที่ 1 vectorc.cpp (ต่อ)

```
  double sum=0.0;
  for(int i=0;i<SIZE; i++)
    sum +=v.c[i]*w.c[i];
    return sum;
}
// Test for equality
int operator== (const vector& v,const vector& w)
{
 for (int i=0;i<SIZE;i++)
   if (v.c[i] !=w.c[i]) return 0;
 return 1;
}
 //Test for inequality
int operator !=(const vector& v,const vector& w)
{
  for (int i=0;i<SIZE; i++)
    if(v.c[i] !=w.c[i]) return 1;
 return 0;
}
 //Addition and assignment
 vector& vector::operator+=(const vector& w)
 {
    for (int i=0;i<SIZE;i++)
      c[i] += w.c[i];
    return *this;
 }
 // subtraction and assignment
 vector& vector::operator-=(const vector& w)
 {
  for(int i=0;i<SIZE; i++)
   c[i]-=w.c[i];
   return *this;
  }
 //Right scalar multiplication and assignment
 vector& vector :: operator *=(double b)
 {
   for (int i=0;i<SIZE;i++)
      c[i] *=b;
    return *this;
  }
 //Subscripting
 double& vector ::operator[] (int i)
 {
  if (i<LOWER ||i>UPPER) {
   cout<<"Subscript out of range\n";
   exit(1);
  }
  return c[i-LOWER];
 }
 //Type conversion from vector to double
 //Yields magnitude of vector
 vector::operator double() const
 {
   double sum=0.0;
  for(int i=0;i<SIZE;i++)
  sum +=c[i]*c[i];
 return sqrt(sum);
 }
 //vector input
 istream& operator>> (istream& ci,vector& v)
 {
   for(int i=0;i<SIZE;i++)
     ci>>v.c[i];
   return ci;
 }
 //Vector output
 ostream& operator<<(ostream& co,const vector& v)
 {
   co<<"(" <<v.c[0];
   for (int i=1;i<SIZE;i++)
    co<<","<<v.c[i];
    co<<")";
    return co;
  }
```

ลิสติงที่ 2 testvc.cpp

```
//File testvc.cpp
//Demonstration program for class vector
#include <iostream.h>
#include "vectorc.h"

double a[SIZE]={1,2,3};
double b[SIZE]={4,5,6};

main()
{
  vector u=a;
  vector v=b;
  vector w;

  cout<<"Enter components for vector w:";
  cin >>w;
  cout <<"\n";

  cout<<"u= "<< u<< "\n";
  cout<<"v= "<< v<< "\n";
  cout<<"w="<< w << "\n";

  cout << "u+v="<< u+v<<"\n";
  cout << "u-v="<<u-v<<"\n";
  cout << "-u=" <<-u <<"\n";
  cout << "2*u="<< 2*u<<"\n";
  cout << "u*3="<< u*3<< "\n";
  cout << "u*v="<< u*v<<"\n\n";

  cout <<"(u==v)="<<(u==v)<<"\n";
  cout <<"(u != v)="<<(u!=v)<<"\n\n";

  cout<<"magnitude u="<<double(u) <<"\n";
  cout << "w[2]="<<w[2]<<"\n\n";

  u +=w;
  v -=w;
  w *=5;
  cout << "After\n u +=w;\n v-=w;\n w *=5;\n\n";
  cout << "u= "<< u<<"\n";
  cout << "v= " << v<<"\n";
  cout << "w= " << w<<"\n";
 }
```

ลิสติงที่ 3 vectorc.h

```
//File vectorc.h
//Header file for class vector

#include <iostream.h>

const LOWER =1;              //LOWER bound for [] operator
const UPPER =3;              //upper bound for [] operator
const SIZE  =UPPER-LOWER+1; //number of components

class vector {
   double c[SIZE];           //array of components
public:
   vector();                 //create null vector
   vector( double* a);       //create vector from array
 //additon subtraction,and multiplication operators
 friend vector operator+ (const vector& v,
                          const vector& w);
 friend vector operator- (const vector& v,
                          const vector& w);
 friend vector operator- (const vector& v);
 friend vector operator* (double a,const vector& w);
 friend vector operator* (const vector& v,double b);
 friend vector operator* (const vector& v,
                          const vector& w);
// Comparison operators
  friend int operator==(const vector& v,
                        const vector& w);
  friend int operator!=(const vector& v,
                        const vector& w);
 //Operation combined with assignment
 vector& operator+= (const vector& w);
 vector& operator-= (const vector& w);
 vector& operator*=(double b);
 double& operator[] (int i);  //subscripting
 operator double() const;     //convert to magnitude
//Insertion and extraction operators

friend istream& operator>> (istream& ci,vector& v);
friend ostream& operator<< (ostream& co,
                            const vector& v);
 };
```